PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

K-center problem*

Andrei Gonçalves R. Massaini and Luiz Fernando C. Rodrigues[1]

Summary

The report aims to compare two algorithmic solutions to the k-center problem, a brute force approach and an approximate approach, where the first would find the best answer, but it is not efficient, can take too long and not be used, and the second will quickly find an answer, however approximate and not exact. The k-center problem is an important issue in combinatorial optimization and has several practical applications.

The report compares the two solutions in terms of execution time, accuracy of the solution found, and scalability for different instance sizes. For this, instances extracted from the OR-library are used. The execution time and accuracy are measured and compared between the two solutions for different values of k and instance sizes.

Overall, the report will provide a detailed and comparative analysis of the two algorithmic solutions to the k-center problem, highlighting their advantages, disadvantages, and performance in different scenarios. This allows readers to better understand the characteristics and trade-offs of solutions, helping to choose the most appropriate approach for different contexts and requirements.

Key words: Graphs, k-problem, k-centers Template. LaTEX. Abakos. Periodicals.

*Article presented to Revista Abakos

[1]Bachelor of Computer Science PUC Minas, Brazil – andrei.massaini@hotmail.com

1. INTRODUCTION

The k-center problem is an important issue in the field of combinatorial optimization. and has applications in various areas of computing, such as logistics, vehicle routing, telecommunications and social network analysis. It involves finding the k points in a graph that minimize the maximum distance between any vertex of the graph and the nearest center.

In this work, our objective was to address the k-center problem in the context of computing, developing two algorithms to solve the problem: an approximate algorithm and another by brute force.

Next, we will provide a detailed analysis of the algorithm implementations, along with directly with a comparison between them, using instances extracted from the library ORlibrary.

2 IMPLEMENTATIONS

2.1 Generating instances

The function *generate_matrix_from_file*, reads a text file that contains information about a graph, including the number of vertices, the number of edges, and the value of the centers. Creates an empty matrix with dimensions corresponding to the number of vertices, initially filling it with infinity. It then fills the main diagonal with zeros to represent that the distance between a vertex and itself is zero.

The function reads each remaining line of the file, which contains information about the edges of the graph, including the connected vertices and the cost of the edge. This information is used to update the matrix by filling the corresponding cells with the edge costs. Since the matrix is symmetric, corresponding cells in opposite positions are also updated.

After creating the initial matrix, the function applies the *Floyd-Warshall* to find the shortest distances between all pairs of vertices in the graph. This algorithm uses three nested loops to iterate over all possible intermediate vertices and update the distances in the matrix. The distance value between two vertices is updated to the minimum between the current value and the sum of the distances passing through an intermediate vertex.

Finally, the function returns the updated distance matrix and the center value as results. The distance matrix represents the minimum distances between all pairs of vertices in the graph, and the center value indicates the vertices considered centers in the graph.

2.2 Algorithms

Algorithm implementations are divided into two main functions:*k_center_brute_approximate* It is *k_center_brute_force*. Both functions receive a distance matrix and a parameter k as input and return their respective found centers, as well as the*radius*of the solution.

2.3 Brute force algorithm

The function*k_center_brute_force*exhaustively searches for the optimal centers in a graph represented by the previously processed distance matrix.

The function generates all possible combinations of K vertices in the graph. For each combination, the greatest distance between a vertex and the vertices of the combination is calculated. The set of vertices that result in the smallest maximum distance are considered as the optimal centers.

When going through all combinations, the function maintains the set of centers with the smallest maximum distance found so far. In the end, it returns this set of centers and the corresponding maximum distance as the result of the function. These values represent the optimal centers found by the exhaustive search in the graph.

When carrying out a complexity analysis, we concluded that this method is unfeasible for very large distances, or with the number k of centers greater than 2, since the algorithm has exponential complexity, as we can see in the following notation: O((num_vertices choose K) * K * num_vertices) That said, we later observed that this method was unable to provide a solution in a timely manner for the given instances, so it becomes necessary to resort to more efficient, approximate approaches, which will be presented below.

2.4 Approximate algorithm

The function*k_center_approximate*implements an approximate algorithm to find the optimal centers in a graph represented by a distance matrix. The algorithm works by selecting the first vertex as the initial center. It then iterates over the remaining vertices and calculates the minimum distance relative to the already selected centers. The vertex with the greatest minimum distance is chosen as a new center. This process is repeated until all desired centers are selected. In the end, the radius of the solution is determined as the greatest distance between a vertex and the furthest center found. This algorithm provides an approximate solution to the problem of optimal centers, being more efficient than the exhaustive search, although the solution obtained may not be optimal. When analyzing the complexity of the method we arrived at the following notation: O(k * num_vertices),

but effective for large instances.

2.5 Comparison:

Next, we will demonstrate the results of the solution radius found by each algorithm, in relation to the real radius contained in the table:

Instance	|V|	k	Ray	Brute force	Approximate Algorithm
1	100	5	127	TIMEOUT	190 (150%)
two	100	10	98	TIMEOUT	132 (135%)
3	100	10	93	TIMEOUT	155 (167%)
4	100	20	74	TIMEOUT	121 (164%)
5	100	33	48	TIMEOUT	72 (150%)
6	200	5	84	TIMEOUT	159 (189%)
7	200	10	64	TIMEOUT	98 (153%)
8	200	20	55	TIMEOUT	83 (151%)
9	200	40	37	TIMEOUT	59 (159%)
10	200	67	20	TIMEOUT	31 (155%)
11	300	5	59	TIMEOUT	82 (139%)
12	300	10	51	TIMEOUT	74 (145%)
13	300	30	35	TIMEOUT	60 (171%)
14	300	60	26	TIMEOUT	41 (158%)
15	300	100	18	TIMEOUT	25 (139%)
16	400	5	47	TIMEOUT	89 (189%)
17	400	10	39	TIMEOUT	57 (146%)
18	400	40	28	TIMEOUT	45 (161%)
19	400	80	18	TIMEOUT	29 (161%)
20	400	133	13	TIMEOUT	19 (146%)
21	500	5	40	TIMEOUT	53 (132%)
22	500	10	38	TIMEOUT	56 (147%)
23	500	50	22	TIMEOUT	34 (155%)
24	500	100	15	TIMEOUT	23 (153%)
25	500	167	11	TIMEOUT	15 (136%)
26	600	5	38	TIMEOUT	50 (132%)
27	600	10	32	TIMEOUT	43 (134%)
28	600	60	18	TIMEOUT	29 (161%)
29	600	120	13	TIMEOUT	20 (154%)
30	600	200	9	TIMEOUT	14 (156%)
31	700	5	30	TIMEOUT	44 (147%)
32	700	10	29	TIMEOUT	46 (159%)
33	700	70	15	TIMEOUT	26 (173%)
34	700	140	11	TIMEOUT	17 (155%)
35	800	5	30	TIMEOUT	38 (127%)
36	800	10	27	TIMEOUT	41 (152%)
37	800	80	15	TIMEOUT	25 (167%)
38	900	5	29	TIMEOUT	39 (134%)
39	900	10	23	TIMEOUT	35 (152%)
40	900	90	13	TIMEOUT	21 (162%)

3 CONCLUSION

By analyzing the results in the table, we can conclude that the brute force algorithm was unable to complete execution on any of the given instances. This occurred because the minimum number of centers in the instances was equal to 5, making the exhaustive search unfeasible in terms of execution time.

On the other hand, the approximate algorithm was able to be executed in all instances. However, we observed a considerable error rate when comparing the found radius with the actual radius of the instances. This percentage difference can be significant in relation to the quality of the solution.

Based on these observations, we can conclude that the brute force algorithm is suitable for small instances where the value of K is less than 3. In such cases, it will likely be able to run smoothly and provide the optimal solution. For instances with a number of centers greater than 3, the use of the approximate algorithm becomes essential, as although it presents a considerable error rate, it is capable of dealing with larger instances in a more efficient and practical way, providing reasonable solutions within an acceptable time.